**Go Programming Language Notes**

**Defer:**

A defer statement defers the execution of a function until the surrounding function returns.

**Os.Exit:**

Note that unlike e.g. C, Go does not use an integer return value from main to indicate exit status. If you’d like to exit with a non-zero status you should use os.Exit.

**Command-line argument:**

<http://en.wikipedia.org/wiki/Command-line_interface#Arguments>

A **command-line argument** or [**parameter**](https://en.wikipedia.org/wiki/Parameter#Computing) is an item of information provided to a program when it is started. A program can have many command-line arguments that identify sources or destinations of information, or that alter the operation of the program.

When a command processor is active a program is typically invoked by typing its name followed by command-line arguments (if any). For example, in [Unix](https://en.wikipedia.org/wiki/Unix) and [Unix-like](https://en.wikipedia.org/wiki/Unix-like) environments, an example of a command-line argument is:

rm file.s

"file.s" is a command-line argument which tells the program [rm](https://en.wikipedia.org/wiki/Rm_(Unix)) to remove the file "file.s".

GO:

[Command-line arguments](http://en.wikipedia.org/wiki/Command-line_interface#Arguments) are a common way to parameterize execution of programs. For example, go run hello.go uses run and hello.go arguments to the go program.

os.Args provides access to raw command-line arguments. Note that the first value in this slice is the path to the program, and os.Args[1:] holds the arguments to the program.

You can get individual args with normal indexing.

**Filters:**

|  |  |
| --- | --- |
| A *line filter* is a common type of program that reads input on stdin, processes it, and then prints some derived result to stdout. grep and sed are common line filters. |  |
| Here’s an example line filter in Go that writes a capitalized version of all input text. You can use this pattern to write your own Go line filters. |  |

[***Command-line flags***](http://en.wikipedia.org/wiki/Command-line_interface#Command-line_option) are a common way to specify options for command-line programs. For example, in wc -l the -l is a command-line flag.

Go provides a flag package supporting basic command-line flag parsing. We’ll use this package to implement our example command-line program.

Basic flag declarations are available for string, integer, and boolean options. Here we declare a string flag word with a default value "foo" and a short description. This flag.String function returns a string pointer (not a string value); we’ll see how to use this pointer below.

**Dynamic Type Declaration / Type Inference in Go**

A dynamic type variable declaration requires the compiler to interpret the type of the variable based on the value passed to it. The compiler does not require a variable to have type statically as a necessary requirement.

Example

Try the following example, where the variables have been declared without any type. Notice, in case of type inference, we initialized the variable y with := operator, whereas x is initialized using = operator.